CORE
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
S. Lemay
Zebra Technologies
H. Tschofenig
ARM Ltd.
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
B. Raymor, Ed.
Microsoft
July 08, 2016

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-03

Abstract

The Constrained Application Protocol (CoAP), although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of the CoAP over UDP protocol includes support for reliable delivery, simple congestion control, and flow control.

Some environments benefit from the availability of CoAP carried over reliable transports such as TCP or TLS. This document outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] or DTLS [RFC6347] over UDP can be used unimpeded. UDP is a good choice for transferring small amounts of data across networks that follow the IP architecture.

Some CoAP deployments need to integrate well with existing enterprise infrastructures, where UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of CoAP usage for IoT can make the use of UDP brittle, resulting in lost or malformed packets.

To address such environments, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. Figure 1 illustrates the layering:

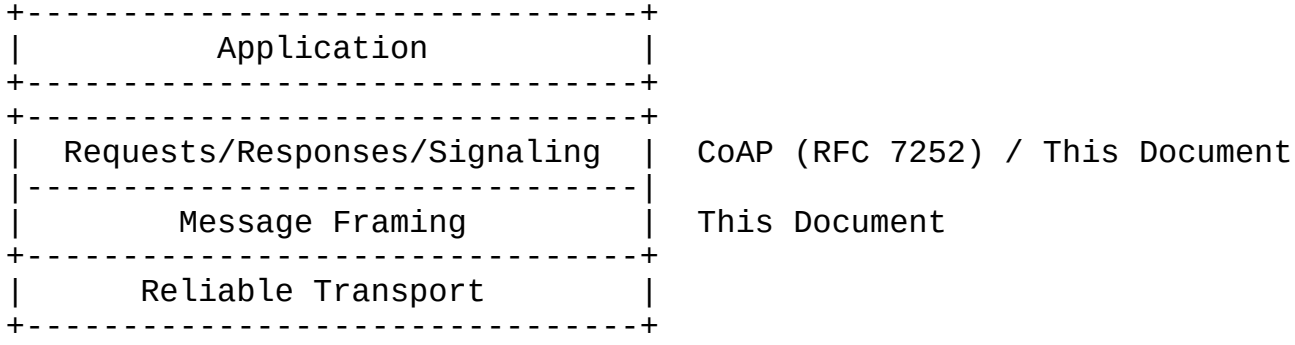

Figure 1: Layering of CoAP over Reliable Transports

Where NATs are present, CoAP over TCP can help with their traversal. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs maintain TCP-based NAT bindings for longer periods based on the assumption that a transport layer protocol, such as TCP, offers additional information about the session life cycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway].

Some environments may also benefit from the ability of TCP to exchange larger payloads, such as firmware images, without application layer segmentation and to utilize the more sophisticated congestion control capabilities provided by many TCP implementations.

CoAP may be integrated into a Web environment where the front-end uses CoAP over UDP from IoT devices to a cloud infrastructure and then CoAP over TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to communicate with the UDP-based IoT device.

To allow IoT devices to better communicate in these demanding environments, CoAP needs to support different transport protocols, namely TCP [RFC0793], in some situations secured by TLS [RFC5246].

In addition, some corporate networks only allow Internet access via a HTTP proxy. In this case, the best transport for CoAP would be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP/1.1 [RFC7230] connection and may be available in an environment that blocks CoAP over UDP. Another scenario for CoAP over WebSockets is a CoAP application running inside a web browser without access to connectivity other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the TCP/TLS and WebSocket protocols. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server accessible over a TCP/TLS or WebSocket connection or via a CoAP intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [RFC7252].

BERT Option:
A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:
The payload of a CoAP message that is affected by a BERT Option in descriptive usage (Section 2.1 of [I-D.ietf-core-block]).

2. CoAP over TCP

The request/response interaction model of CoAP over TCP is the same as CoAP over UDP. The primary differences are in the message layer. CoAP over UDP supports optional reliability by defining four types of messages: Confirmable, Non-confirmable, Acknowledgement, and Reset. TCP eliminates the need for the message layer to support reliability. As a result, message types are not defined in CoAP over TCP.

2.1. Messaging Model

Conceptually, CoAP over TCP replaces most of the CoAP over UDP message layer with a framing mechanism on top of the byte stream provided by TCP/TLS, conveying the length information for each message that on datagram transports is provided by the UDP/DTLS datagram layer.

TCP ensures reliable message transmission, so the CoAP over TCP messaging layer is not required to support acknowledgements or detection of duplicate messages. As a result, both the Type and Message ID fields are no longer required and are removed from the CoAP over TCP message format. All messages are also untyped.

Figure 2 illustrates the difference between CoAP over UDP and CoAP over reliable transport. The removed Type (no type) and Message ID fields are indicated by dashes.

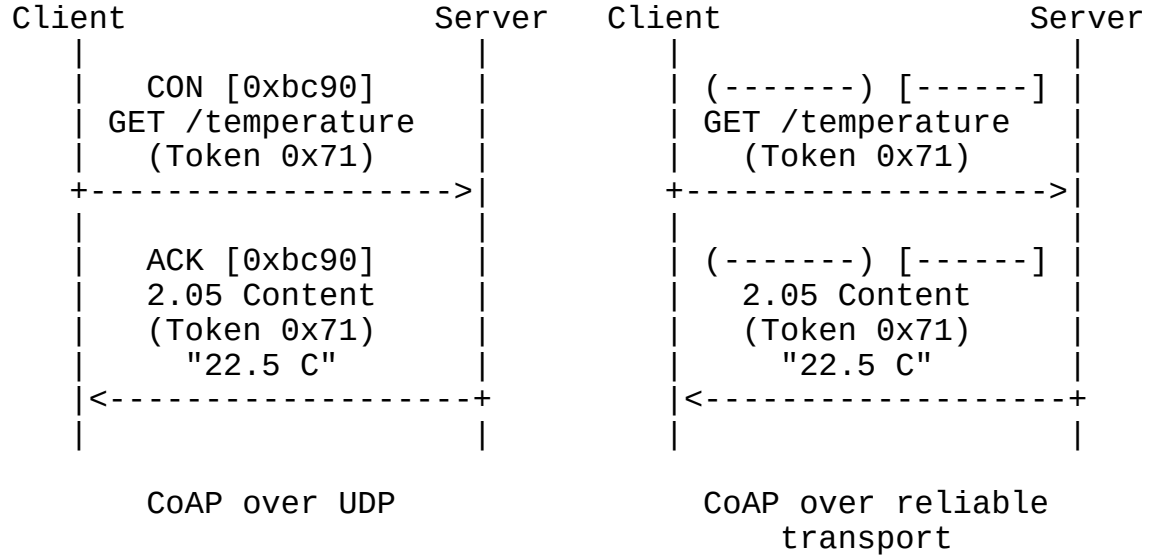

Figure 2: Comparison between CoAP over unreliable and reliable transport.

2.2. UDP-to-TCP gateways

A UDP-to-TCP gateway MUST discard all Empty messages (Code 0.00) after processing at the message layer. For Confirmable (CON), Non-Confirmable (NOM), and Acknowledgement (ACK) messages that are not Empty, their contents are repackaged into untyped messages.

2.3. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 3, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate and for providing length information.

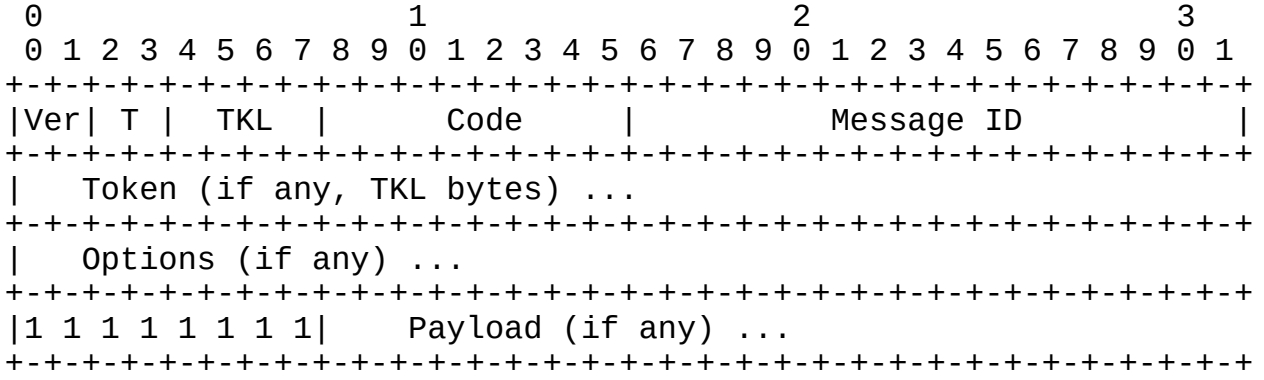

Figure 3: RFC 7252 defined CoAP Message Format.

The CoAP over TCP message format is very similar to the format specified for CoAP over UDP. The differences are as follows:

- Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The "T" and "Message ID" fields in the CoAP message header are elided.

- The "Ver" field is elided as well. In constrast to the UDP message layer for UDP and DTLS, the CoAP over TCP message layer does not send a version number in each message. If required in the future, a new Capability and Settings Option (See Appendix A) could be defined to support version negotiation.

- In a stream oriented transport protocol such as TCP, a form of message delimitation is needed. For this purpose, CoAP over TCP introduces a length field with variable size. Figure 4 shows the adjusted CoAP message format with a modified structure for the fixed header (first 4 bytes of the CoAP over UDP header), which includes the length information of variable size, shown here as an 8-bit length.

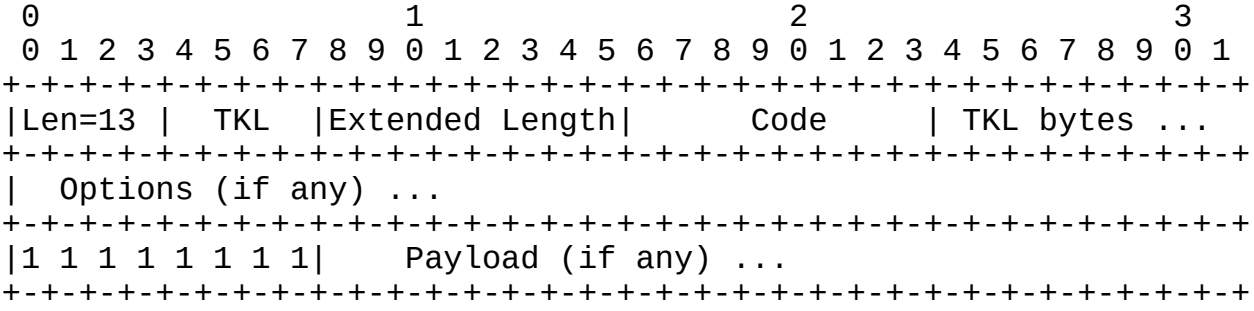

Figure 4: CoAP frame with 8-bit Extended Length field.

Length (Len): 4-bit unsigned integer. A value between 0 and 12 directly indicates the length of the message in bytes starting with the first bit of the Options field. Three values are reserved for special constructs:

13: An 8-bit unsigned integer (Extended Length) follows the initial byte and indicates the length of options/payload minus 13.

14: A 16-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 269.

15: A 32-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 65805.

The encoding of the Length field is modeled on CoAP Options (see section 3.1 of [RFC7252]).

The following figures show the message format for the 0-bit, 16-bit, and the 32-bit variable length cases.

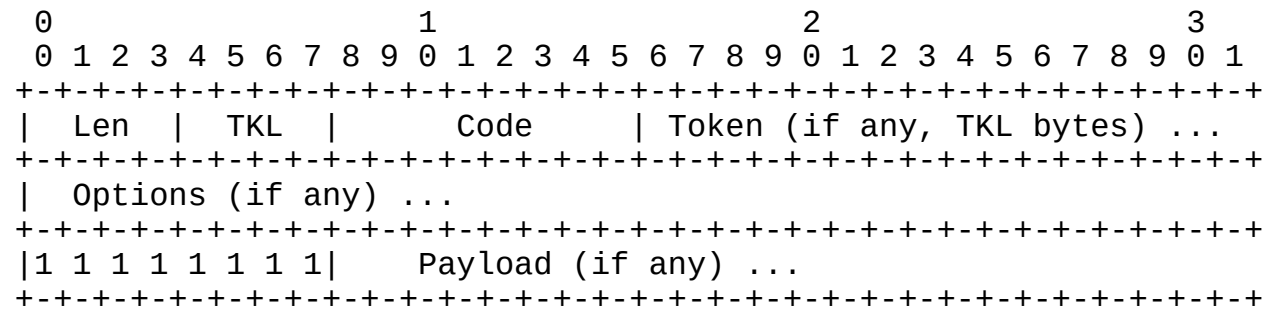

Figure 5: CoAP message format without an Extended Length field.

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload would be encoded as shown in Figure 6.

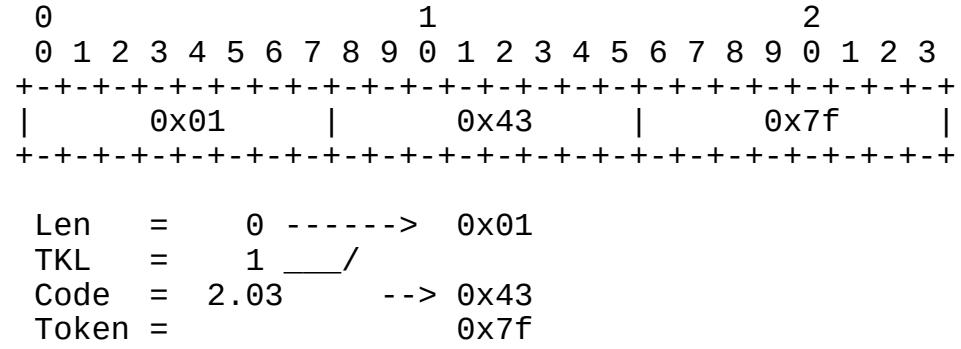

Figure 6: CoAP message with no options or payload.

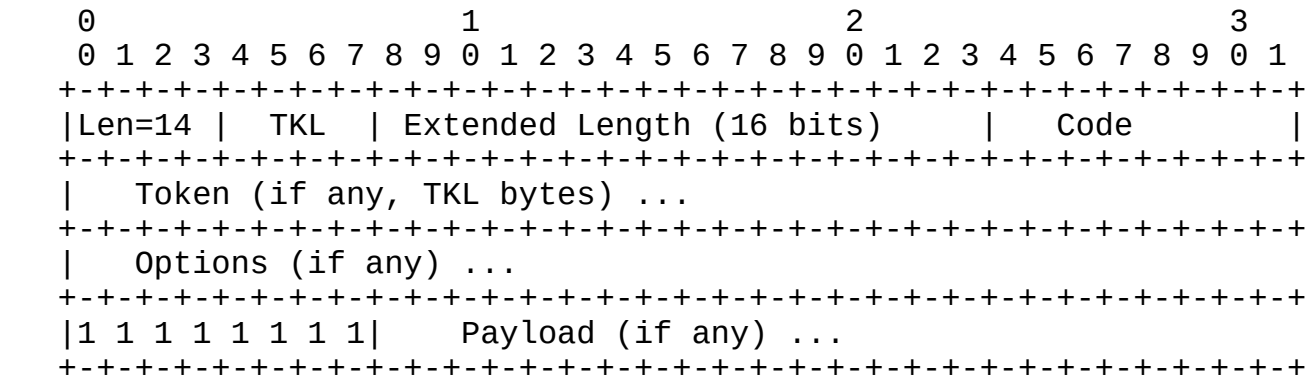

Figure 7: CoAP message format with 16-bit Extended Length field.

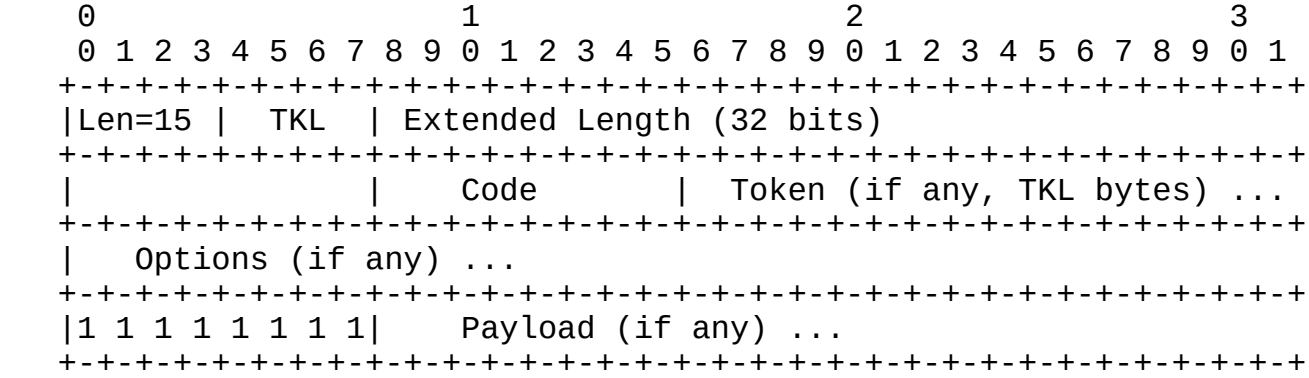

Figure 8: CoAP message format with 32-bit Extended Length field.

The semantics of the other CoAP header fields are left unchanged.

2.4. Message Transmission

CoAP requests and responses are exchanged asynchronously over the TCP/TLS connection. A CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the TCP/TLS protocol.

Since the TCP protocol provides ordered delivery of messages, the mechanism for reordering detection when observing resources [RFC7641]

is not needed. The value of the Observe Option in notifications MAY be empty on transmission and MUST be ignored on reception.

3. CoAP over WebSockets

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client retrieving or updating a CoAP resource located at a CoAP server that exposes a WebSocket endpoint (Figure 9). The CoAP client acts as the WebSocket client, establishes a WebSocket connection, and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket connection can be used for any number of requests.

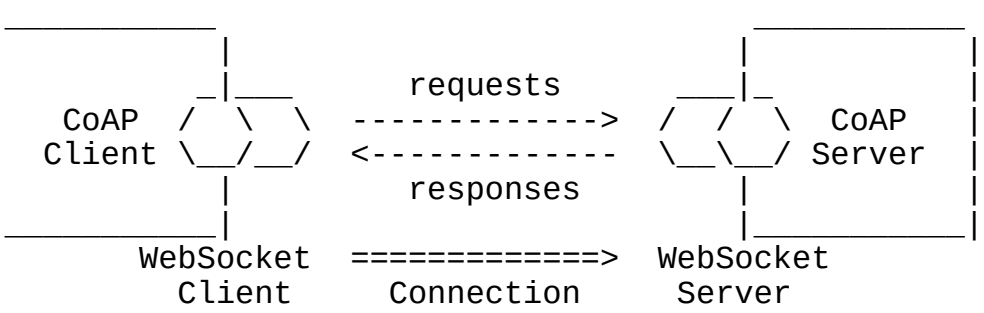

Figure 9: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

The challenge with this configuration is how to identify a resource in the namespace of the CoAP server. When the WebSocket protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. This means it is necessary for the client to identify both the WebSocket endpoint (identified by a "ws" or "wss" URI) and the path and query of the CoAP resource within that endpoint from the same URI. When the WebSocket protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

Section 6.2 defines a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

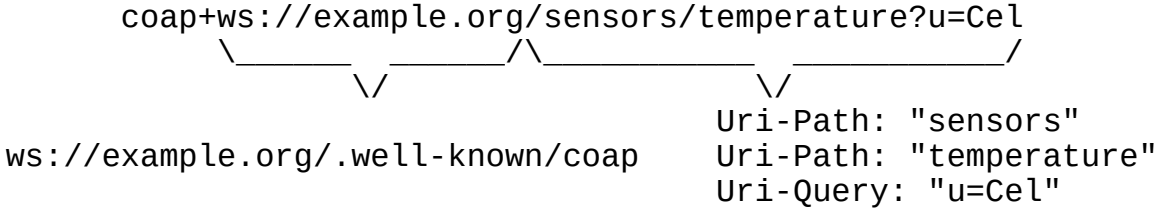

Figure 10: The "coap+ws" URI Scheme

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 11), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The client specifies the resource to be updated or retrieved in the Proxy-URI Option.

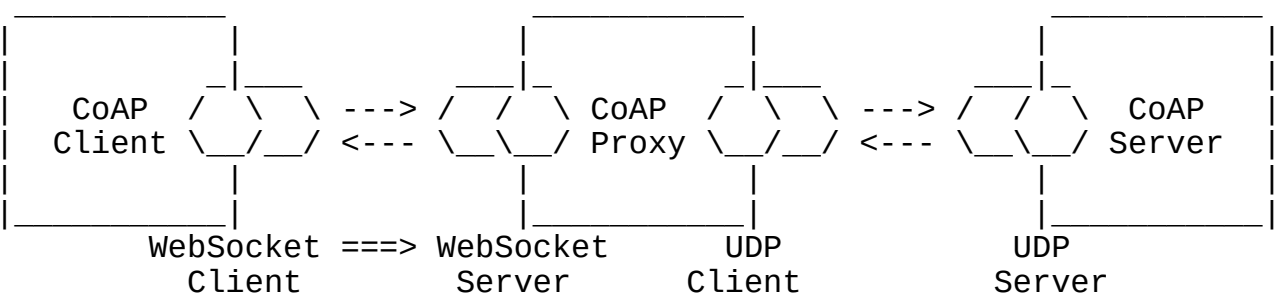

Figure 11: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 12). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is unreachable. Because the WebSocket server is the only way to reach the CoAP server, the CoAP proxy should be a Reverse Proxy.

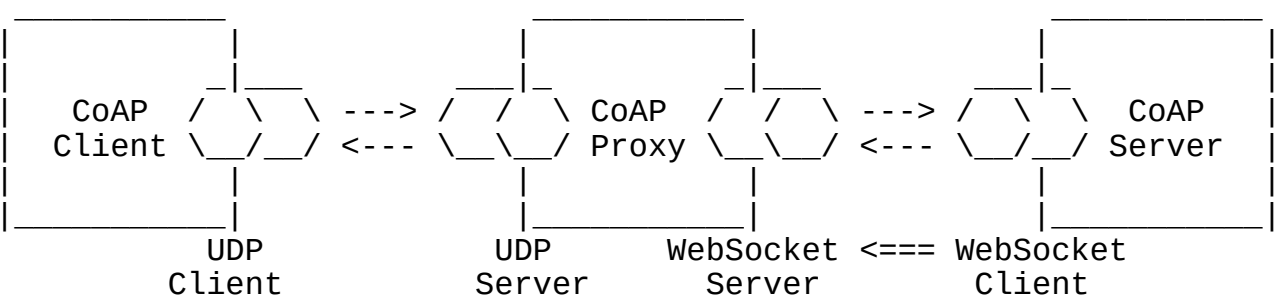

Figure 12: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

Further configurations are possible, including those where a WebSocket connection is established through an HTTP proxy.

CoAP over WebSockets is intentionally very similar to CoAP over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [RFC7252].

3.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 13 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document. Any later, incompatible versions of CoAP or CoAP over WebSockets will use a different subprotocol name.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host header field also indicates the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server.

```
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap
```

Figure 13: Example of an Opening Handshake

3.2. Message Format

Once a WebSocket connection is established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format shown in Figure 14 is the same as the CoAP over TCP message format (see Section 2.3) with one restriction. The Length (Len) field MUST be set to zero because the WebSockets frame contains the length.

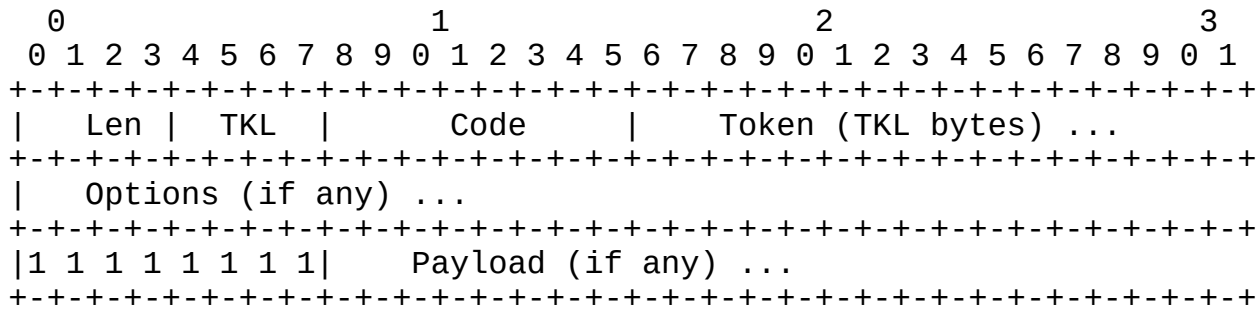

Figure 14: CoAP Message Format over WebSockets

The CoAP over TCP message format eliminates the Version field defined in CoAP over UDP. If CoAP version negotiation is required in the future, CoAP over WebSockets can address the requirement by the definition of a new subprotocol identifier that is negotiated during the opening handshake.

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing. If it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a block-wise fashion as defined in [I-D.ietf-core-block].

Empty messages (Code 0.00) MUST be ignored by the recipient (see also Section 4.4).

3.3. Message Transmission

CoAP requests and responses are exchanged asynchronously over the WebSocket connection. A CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the WebSocket protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

Since the WebSocket protocol provides ordered delivery of messages, the mechanism for reordering detection when observing resources [RFC7641] is not needed. The value of the Observe Option in notifications MAY be empty on transmission and MUST be ignored on reception.

3.4. Connection Health

When a client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it.

To check the health of the WebSocket connection (and thereby of all active requests, if any), the client can send a Ping frame or an unsolicited Pong frame as specified in Section 5.5 of [RFC6455]. There is no way to retransmit a request without creating a new one. Re-registering interest in a resource is permitted, but entirely unnecessary.

3.5. Closing the Connection

The WebSocket connection is closed as specified in Section 7 of [RFC6455].

All requests for which the CoAP client has not received a response yet are cancelled when the connection is closed. If the client observes one or more resources over the WebSocket connection, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client from the lists of observers when the connection is closed.

4. Signaling

Signaling messages are introduced to allow peers to:

- Share characteristics such as maximum message size for the connection
- Shutdown the connection in an ordered fashion
- Terminate the connection in response to a serious error condition

Signaling is a third basic kind of message in CoAP, after requests and responses. Signaling messages share a common structure with the

existing CoAP messages. There is a code, a token, options, and an optional payload.

(See Section 3 of [RFC7252] for the overall structure, as adapted to the specific transport.)

4.1. Signaling Codes

A code in the 7.01-7.31 range indicates a Signaling message. Values in this range are assigned by the "CoAP Signaling Codes" sub-registry (see Section 8.1).

For each message, there is a sender and a peer receiving the message.

Payloads in Signaling messages are diagnostic payloads (see Section 5.5.2 of [RFC7252]), unless otherwise defined by a Signaling message option.

4.2. Signaling Option Numbers

Option numbers for Signaling messages are specific to the message code. They do not share the number space with CoAP options for request/response messages or with Signaling messages using other codes.

Option numbers are assigned by the "CoAP Signaling Option Numbers" sub-registry (see Section 8.2).

Signaling options are elective or critical as defined in Section 5.4.1 of [RFC7252]). If a Signaling option is critical and not understood by the receiver, it MUST abort the connection (see Section 4.6). If the option is understood but cannot be processed, the option documents the behavior.

4.3. Capability and Settings Messages (CSM)

Capability and Settings messages (CSM) are used for two purposes:

- Each capability option advertises one capability of the sender to the recipient.

- Setting options indicate a setting that will be applied by the sender.

Most CSM Options are useful mainly as initial messages in the connection.

Both capability and settings options are cumulative. A Capability and Settings message does not invalidate a previously sent capability indication or setting even if it is not repeated. A capability message without any option is a no-operation (and can be used as such). An option that is sent might override a previous value for the same option. The option defines how to handle this case if needed.

Base values are listed below for CSM Options. These are the values for the Capability and Setting before any Capability and Settings messages sends a modified value.

These are not default values for the option as defined in Section 5.4.4 in [RFC7252]. A default value would mean that an empty Capability and Settings message would result in the option being set to its default value.

Capability and Settings messages are indicated by the 7.01 code (CSM).

4.3.1. Server-Name Setting Option

Number	Applies to	Name	Format	Length	Base Value
1	CSM	Server-Name	string	1-255	(see below)

A client can use the Server-Name critical option to indicate the default value for the Uri-Host Options in the messages that it sends to the server. It has the same restrictions as the Uri-Host Option (Section 5.10 of [RFC7252].

For TLS, the initial value for the Server-Name Option is given by the SNI value.

For Websockets, the initial value for the Server-Name Option is given by the HTTP Host header field.

4.3.2. Max-Message-Size Capability Option

The sender can use the Max-Message-Size elective option to indicate the maximum message size in bytes that it can receive.

Number	Applies to	Name	Format	Length	Base Value
2	CSM	Max-Message-Size	uint	0-4	1152

As per Section 4.6 of [RFC7252], the base value (and the value used when this option is not implemented) is 1152. A peer that relies on this option being indicated with a certain minimum value will enjoy limited interoperability.

4.3.3. Block-wise Transfer Capability Option

Number	Applies to	Name	Format	Length	Base Value
4	CSM	Block-wise Transfer	empty	0	(none)

A sender can use the Block-wise Transfer elective Option to indicate that it supports the block-wise transfer protocol [I-D.ietf-core-block].

If the option is not given, the peer has no information about whether block-wise transfers are supported by the sender or not. An implementation that supports block-wise transfers SHOULD indicate the Block-wise Transfer Option. If a Max-Message-Size Option is indicated with a value that is greater than 1152 (in the same or a different CSM message), the Block-wise Transfer Option also indicates support for BERT (see Section 5).

4.4. Ping and Pong Messages

In CoAP over TCP, Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient (see also Section 4.4). This provides a basic keep-alive function that can refresh NAT bindings. In contrast, Ping and Pong messages are a bidirectional exchange.

Upon receipt of a Ping message, a single Pong message is returned with the identical token. As with all Signaling messages, the recipient of a Ping or Pong message MUST ignore elective options it does not understand.

Ping and Pong messages are indicated by the 7.02 code (Ping) and the 7.03 code (Pong).

4.4.1. Custody Option

Number	Applies to	Name	Format	Length	Base Value
2	Ping, Pong	Custody	empty	0	(none)

A peer replying to a Ping message can add a Custody elective option to the Pong message it returns. This option indicates that the application has processed all request/response messages that it has received in the present connection ahead of the Ping message that prompted the Pong message. (Note that there is no definition of specific application semantics of "processed", but there is an expectation that the sender of the Ping leading to the Pong with a Custody Option should be able to free buffers based on this indication.)

A Custody elective option can also be sent in a Ping message to explicitly request the return of a Custody Option in the Pong message. A peer is always free to indicate that it has finished processing all previous request/response messages by sending a Custody Option in a Pong message. A peer is also free NOT to send a Custody Option in case it is still processing previous request/response messages; however, it SHOULD delay its response to a Ping with a Custody Option until it can also return one.

4.5. Release Messages

A release message indicates that the sender does not want to continue maintaining the connection and opts for an orderly shutdown; the details are in the options. A diagnostic payload MAY be included. A release message will normally be replied to by the peer by closing the TCP/TLS connection. Messages may be in flight when the sender decides to send a Release message. The general expectation is that these will still be processed.

Release messages are indicated by the 7.04 code (Release).

Release messages can indicate one or more reasons using elective options. The following options are defined:

Number	Applies to	Name	Format	Length	Base Value
2	Release	Bad-Server-Name	empty	0	(none)

The Bad-Server-Name elective option indicates that the default indicated by the CSM Server-Name Option is unlikely to be useful for this server.

Number	Applies to	Name	Format	Length	Base Value
4	Release	Alternate-Address	string	1-255	(none)

The Alternative-Address elective option requests the peer to instead open a connection of the same kind as the present connection to the alternative transport address given. Its value is in the form "authority" as defined in Section 3.2 of [RFC3986].

Number	Applies to	Name	Format	Length	Base Value
6	Release	Hold-Off	uint	0-3	(none)

The Hold-Off elective option indicates that the server is requesting that the peer not reconnect to it for the number of seconds given in the value.

4.6. Abort Messages

An abort message indicates that the sender is unable to continue maintaining the connection and cannot even wait for an orderly release. The sender shuts down the connection immediately after the abort (and may or may not wait for a release or abort message or connection shutdown in the inverse direction). A diagnostic payload SHOULD be included in the Abort message. Messages may be in flight when the sender decides to send an abort message. The general expectation is that these will NOT be processed.

Abort messages are indicated by the 7.05 code (Abort).

Abort messages can indicate one or more reasons using elective options. The following option is defined:

Number	Applies to	Name	Format	Length	Base Value
2	Abort	Bad-CSM-Option	uint	0-2	(none)

The Bad-CSM-Option Option indicates that the sender is unable to process the CSM option identified by its option number, e.g. when it is critical and the option number is unknown by the sender, or when there is parameter problem with the value of an elective option. More detailed information SHOULD be included as a diagnostic payload.

One reason for an sender to generate an abort message is a general syntax error in the byte stream received. No specific option has been defined for this, as the details of that syntax error are best left to a diagnostic payload.

4.7. Capability and Settings examples

An encoded example of a Ping message with a non-empty token is shown in Figure 15.

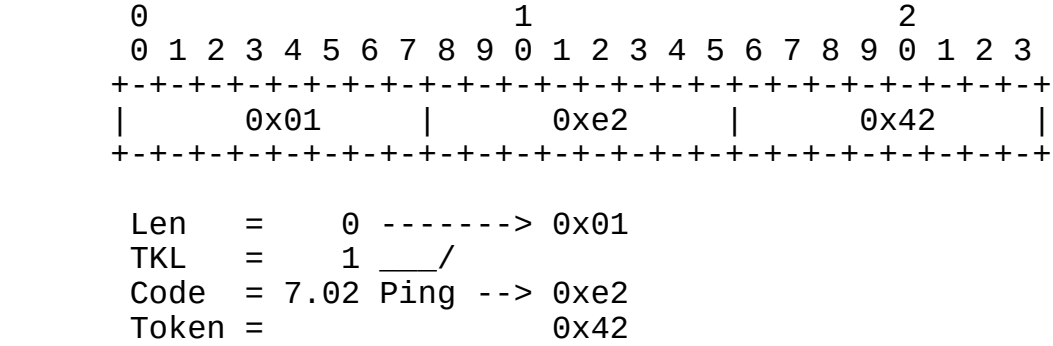

Figure 15: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 16.

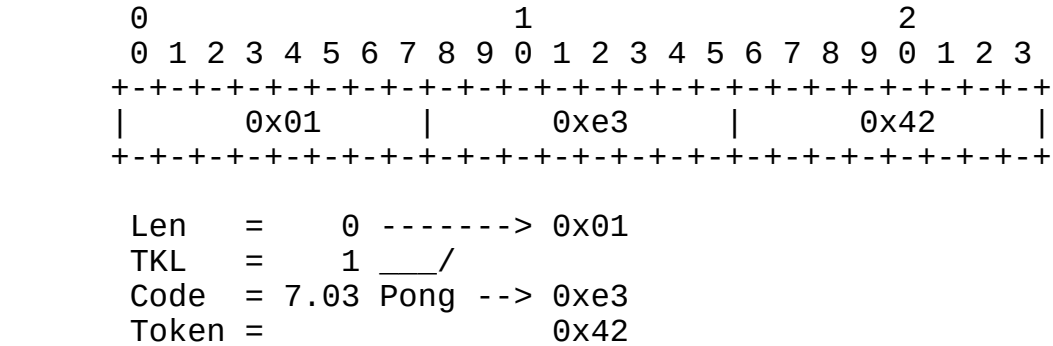

Figure 16: Pong Message Example

5. Block-wise Transfer and Reliable Transports

The message size restrictions defined in Section 4.6 of CoAP [RFC7252] to avoid IP fragmentation are not necessary when CoAP is used over a reliable byte stream transport. While this suggests that the Block-wise transfer protocol [I-D.ietf-core-block] is also no longer needed, it remains applicable for a number of cases:

- large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single TCP connection is used
- a UDP-to-TCP gateway may simply not have the context to convert a message with a Block Option into the equivalent exchange without any use of a Block Option (it would need to convert the entire blockwise exchange from start to end into a single exchange)

The ’Block-wise Extension for Reliable Transport (BERT)’ extends the Block protocol to enable the use of larger messages over a reliable transport.

The use of this new extension is signaled by sending Block1 or Block2 Options with SZX == 7 (a "BERT option"). SZX == 7 is a reserved value in [I-D.ietf-core-block].

In control usage, a BERT option is interpreted in the same way as the equivalent Option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic Block protocol, the recipient of a CoAP request with a BERT option in control usage is allowed to respond with a different SZX value, e.g. to send a non-BERT block instead.

In descriptive usage, a BERT Option is interpreted in the same way as the equivalent Option with SZX == 6, except that the payload is allowed to contain a multiple of 1024 bytes (non-final BERT block) or more than 1024 bytes (final BERT block).

The recipient of a non-final BERT block (M=1) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers starting at, and sequentially increasing from, the block number given in the Block Option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number with 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with SZX == 6, the recipient of a final BERT block (M=0) simply appends the payload at the byte position that is indicated by the block number multiplied with 1024.

The following examples illustrate BERT options. A value of SZX == 7 is labeled as "BERT" or as "BERT(nnn)" to indicate a payload of size nnn.

In all these examples, a Block Option is decomposed to indicate the kind of Block Option (1 or 2) followed by a colon, the block number (NUM), more bit (M), and block size exponent (2**(SZX+4)) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

5.1. Example: GET with BERT Blocks

Figure 17 shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block number increments to move the position inside the response body forward.

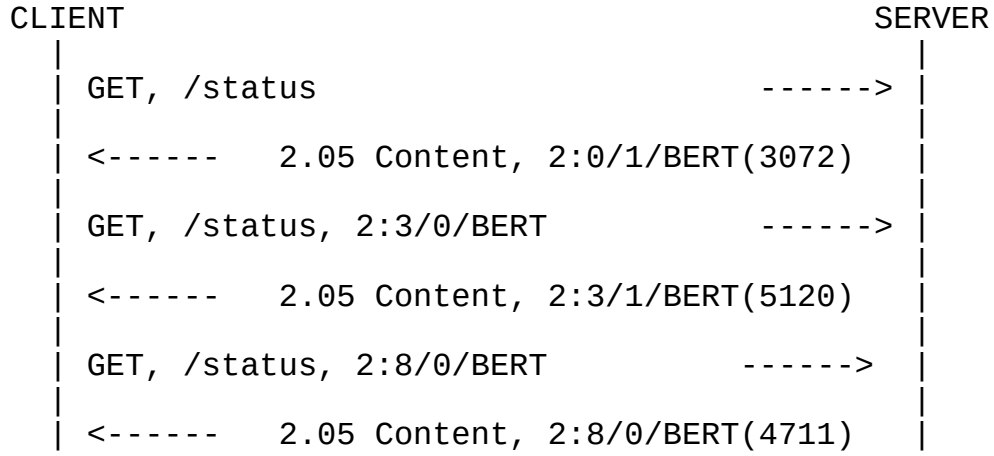

Figure 17: GET with BERT blocks.

5.2. Example: PUT with BERT Blocks

Figure 18 demonstrates a PUT exchange with BERT blocks.

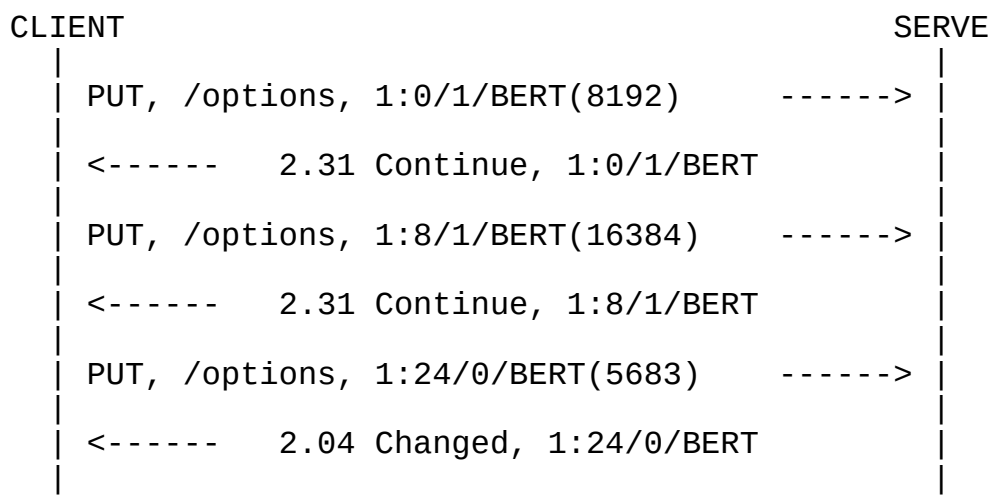

Figure 18: PUT with BERT blocks.

6. CoAP URIs

CoAP over UDP [RFC7252] defines the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource.

6.1. CoAP over TCP and TLS URIs

CoAP over TCP uses the "coap+tcp" URI scheme. CoAP over TLS uses the "coaps+tcp" scheme. The rules from Section 6 of [RFC7252] apply to both of these URI schemes.

[RFC7252], Section 8 (Multicast CoAP) is not applicable to these schemes.

Resources made available via one of the "coap+tcp" or "coaps+tcp" schemes have no shared identity with the other scheme or with the "coap" or "coaps" scheme, even if their resource identifiers indicate the same authority (the same host listening to the same port). The schemes constitute distinct namespaces and, in combination with the authority, are considered to be distinct origin servers.

6.1.1. coap+tcp URI scheme

```
coap-tcp-URI = "coap+tcp:" "//" host [ ":" port ] path-abempty
               [ "?" query ]
```

The semantics defined in [RFC7252], Section 6.1, apply to this URI scheme, with the following changes:

- The port subcomponent indicates the TCP port at which the CoAP server is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

6.1.2. coaps+tcp URI scheme

```
coaps-tcp-URI = "coaps+tcp:" "//" host [ ":" port ] path-abempty
                [ "?" query ]
```

The semantics defined in [RFC7252], Section 6.2, apply to this URI scheme, with the following changes:

- The port subcomponent indicates the TCP port at which the TLS server for the CoAP server is located. If it is empty or not given, then the default port 443 is assumed (this is different from the default port for "coaps", i.e., CoAP over DTLS over UDP).

- When CoAP is exchanged over TLS port 443, the "TLS Application Layer Protocol Negotiation Extension" [RFC7301] MUST be used to allow demultiplexing at the server-side.

6.2. CoAP over WebSockets URIs

For the first configuration discussed in Section 3, this document defines two new URIs schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint.

The WebSocket endpoint is identified by a "ws" or "wss" URI that is composed of the authority part of the "coap+ws" or "coap+wss" URI, respectively, and the well-known path "/.well-known/coap" [RFC5785]. The path and query parts of a "coap+ws" or "coap+wss" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

```
coap-ws-URI =
   "coap+ws:" "//" host [ ":" port ] path-abempty [ "?" query ]

coap-wss-URI =
   "coap+wss:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in the URI options of a CoAP request.

6.2.1. Decomposing and Composing URIs

The steps for decomposing a "coap+ws" or "coap+wss" URI into CoAP options are the same as specified in Section 6.4 of [RFC7252] with the following changes:

- The <scheme> component MUST be "coap+ws" or "coap+wss" when converted to ASCII lowercase.
- A Uri-Host Option MUST only be included in a request when the <host> component does not equal the uri-host component in the Host header field in the WebSocket handshake.
- A Uri-Port Option MUST only be included in a request if |port| does not equal the port component in the Host header field in the WebSocket handshake.

The steps to construct a URI from a request’s options are changed accordingly.

7. Security Considerations

The security considerations of [RFC7252] apply.

Implementations of CoAP MUST use TLS version 1.2 or higher for CoAP over TLS. The general TLS usage guidance in [RFC7525] SHOULD be followed.

Guidelines for use of cipher suites and TLS extensions can be found in [I-D.ietf-dice-profile].

TLS does not protect the TCP header. This may, for example, allow an on-path adversary to terminate a TCP connection prematurely by spoofing a TCP reset message.

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [RFC7252]. The security considerations of [RFC6455] apply.

7.1. Signaling Messages

- The guidance given by an Alternative-Address Option cannot be followed blindly. In particular, a peer MUST NOT assume that a successful connection to the Alternative-Address inherits all the security properties of the current connection.

- SNI vs. Server-Name: Any security negotiated in the TLS handshake is for the SNI name exchanged in the TLS handshake and checked against the certificate provided by the server. The Server-Name Option cannot be used to extend these security properties to the additional server name.

8. IANA Considerations

8.1. Signaling Codes

IANA is requested to create a third sub-registry for values of the Code field in the CoAP header (Section 12.1 of [RFC7252]). The name of this sub-registry is "CoAP Signaling Codes".

Each entry in the sub-registry must include the Signaling Code in the range 7.01-7.31, its name, and a reference to its documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
7.01	CSM	[RFCthis]
7.02	Ping	[RFCthis]
7.03	Pong	[RFCthis]
7.04	Release	[RFCthis]
7.05	Abort	[RFCthis]

Table 1: CoAP Signal Codes

All other Signaling Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC5226].

8.2. CoAP Signaling Option Numbers Registry

IANA is requested to create a sub-registry for signaling options similar to the CoAP Option Numbers Registry (Section 12.2 of [RFC7252]), with the single change that a fourth column is added to the sub-registry that is one of the codes in the Signaling Codes subregistry (Section 8.1).

The name of this sub-registry is "CoAP Signaling Option Numbers".

Initial entries in this sub-registry are as follows:

Number	Applies to	Name	Reference
1	CSM	Server-Name	[RFCthis]
2	CSM	Max-Message-Size	[RFCthis]
4	CSM	Block-wise-Transfer	[RFCthis]
2	Ping, Pong	Custody	[RFCthis]
2	Release	Bad-Server-Name	[RFCthis]
4	Release	Alternative-Address	[RFCthis]
6	Release	Hold-Off	[RFCthis]
2	Abort	Bad-CSM-Option	[RFCthis]

Table 2: CoAP Signal Option Codes

The IANA policy for future additions to this sub-registry is based on number ranges for the option numbers, analogous to the policy defined in Section 12.2 of [RFC7252].

8.3. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
coap+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFCthis]

Port Number.
5683

Similarly, IANA is requested to assign the service name "coaps+tcp", in accordance with [RFC6335]. However, no separate port number is used for "coaps" over TCP; instead, the ALPN protocol ID defined in Section 8.6 is used over port 443.

Service Name.
coaps+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFC7301], [RFCthis]

Port Number.
443 (see also Section 8.6 of [RFCthis]})

8.4. URI Scheme Registration

This document registers two new URI schemes, namely "coap+tcp" and "coaps+tcp", for the use of CoAP over TCP and for CoAP over TLS over TCP, respectively. The "coap+tcp" and "coaps+tcp" URI schemes can thus be compared to the "http" and "https" URI schemes.

The syntax of the "coap" and "coaps" URI schemes is specified in Section 6 of [RFC7252] and the present document re-uses their semantics for "coap+tcp" and "coaps+tcp", respectively, with the exception that TCP, or TLS over TCP is used as a transport protocol.

IANA is requested to add these new URI schemes to the registry established with [RFC7595].

8.4.1. coap+ws

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws". The registration request complies with [RFC4395].

URL scheme name.
coap+ws

Status.
Permanent

URI scheme syntax.
Defined in Section N of [RFCthis]

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket protocol.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.
None.

Security Considerations.

See Section N of [RFCthis]

Contact.
IETF chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCthis]

8.4.2. coap+wss

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss". The registration request complies with [RFC4395].

URL scheme name.
coap+wss

Status.
Permanent

URI scheme syntax.
Defined in Section N of [RFCthis]

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket protocol secured with Transport Layer Security (TLS).

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.
None.

Security Considerations.
See Section N of [RFCthis]

Contact.

IETF chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCthis]

8.5. Well-Known URI Suffix Registration

IANA is requested to register the ’coap’ well-known URI in the "Well-Known URIs" registry. This registration request complies with [RFC5785]:

URI Suffix.
coap

Change controller.
IETF

Specification document(s).
[RFCthis]

Related information.
None.

8.6. ALPN Protocol ID

IANA is requested to assign the following value in the registry "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]:

Protocol.
CoAP

Identification Sequence.
0x63 0x6f 0x61 0x70 ("coap")

Reference.
[RFCthis]

8.7. WebSocket Subprotocol Registration

IANA is requested to register the WebSocket CoAP subprotocol under the "WebSocket Subprotocol Name Registry":

Subprotocol Identifier.
coap

Subprotocol Common Name.
Constrained Application Protocol (CoAP)

Subprotocol Definition.
[RFCthis]

9. References

9.1. Normative References

[I-D.ietf-dice-profile]
Tschofenig, H. and T. Fossati, "TLS/DTLS Profiles for the Internet of Things", draft-ietf-dice-profile-17 (work in progress), October 2015.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", RFC 4395, DOI 10.17487/RFC4395, February 2006, <http://www.rfc-editor.org/info/rfc4395>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <http://www.rfc-editor.org/info/rfc7595>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

9.2. Informative References

[HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement, 2010.

[I-D.becker-core-coap-sms-gprs] Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.

[I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-20 (work in progress), April 2016.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <http://www.rfc-editor.org/info/rfc6335>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Appendix A. Negotiating Protocol Versions

CoAP is defined in [RFC7252] with a version number of 1. At this time, there is no known reason to support version numbers different from 1.

In contrast to the message layer for UDP and DTLS, the CoAP over TCP message format does not include a version number. If version negotiation needs to be addressed in the future, then Capability and Settings have been specifically designed to enable such a potential feature.

Appendix B. CoAP over WebSocket Examples

This section gives examples for the first two configurations discussed in Section 3.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be

as follows. Figure 19 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.

2. It establishes a WebSocket connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.

3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.

4. It waits for the server to return a response.

5. The CoAP client uses the connection for further requests, or the connection is closed.

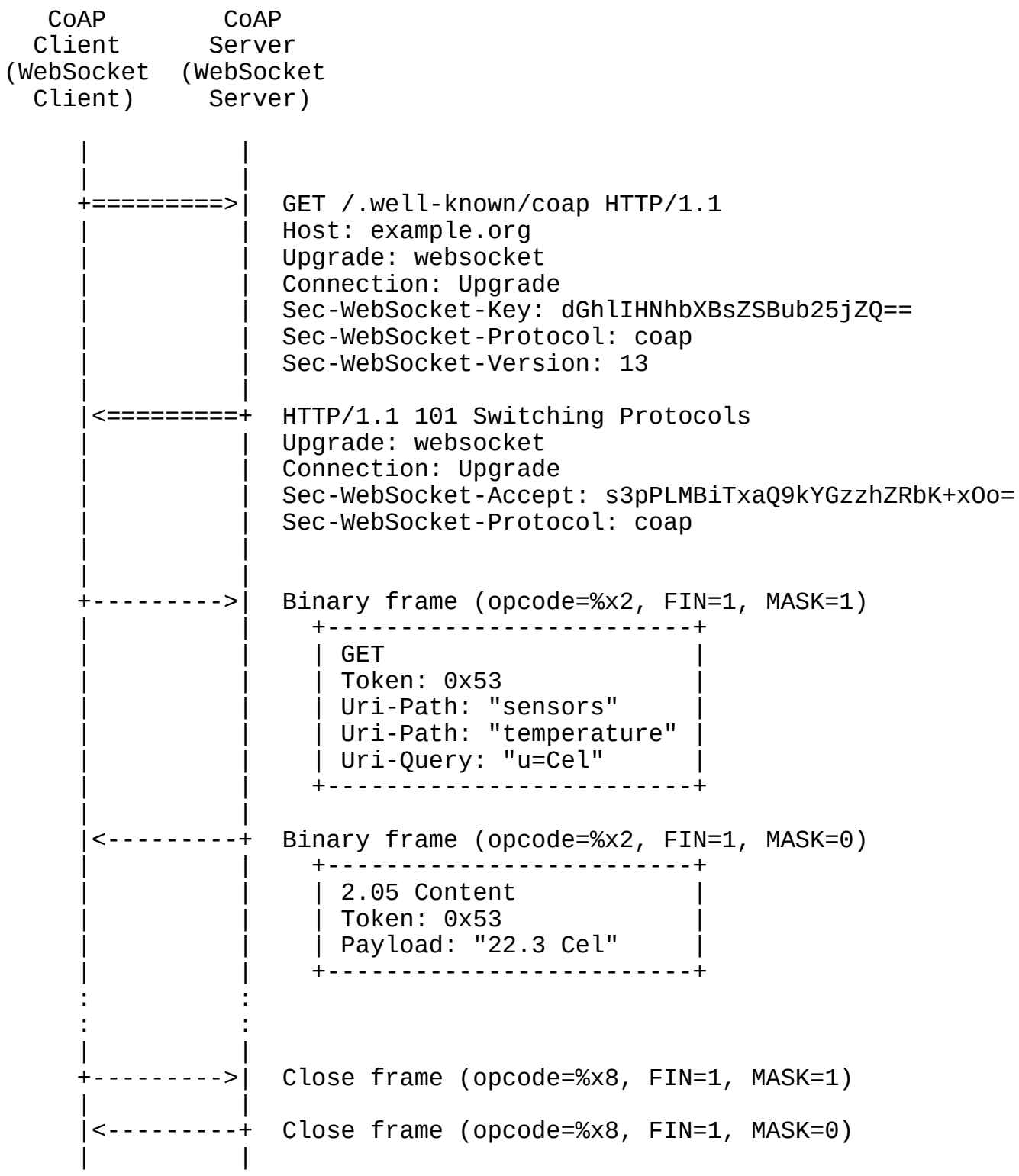

Figure 19: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 20 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource "coap://[2001:DB8::1]/". The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

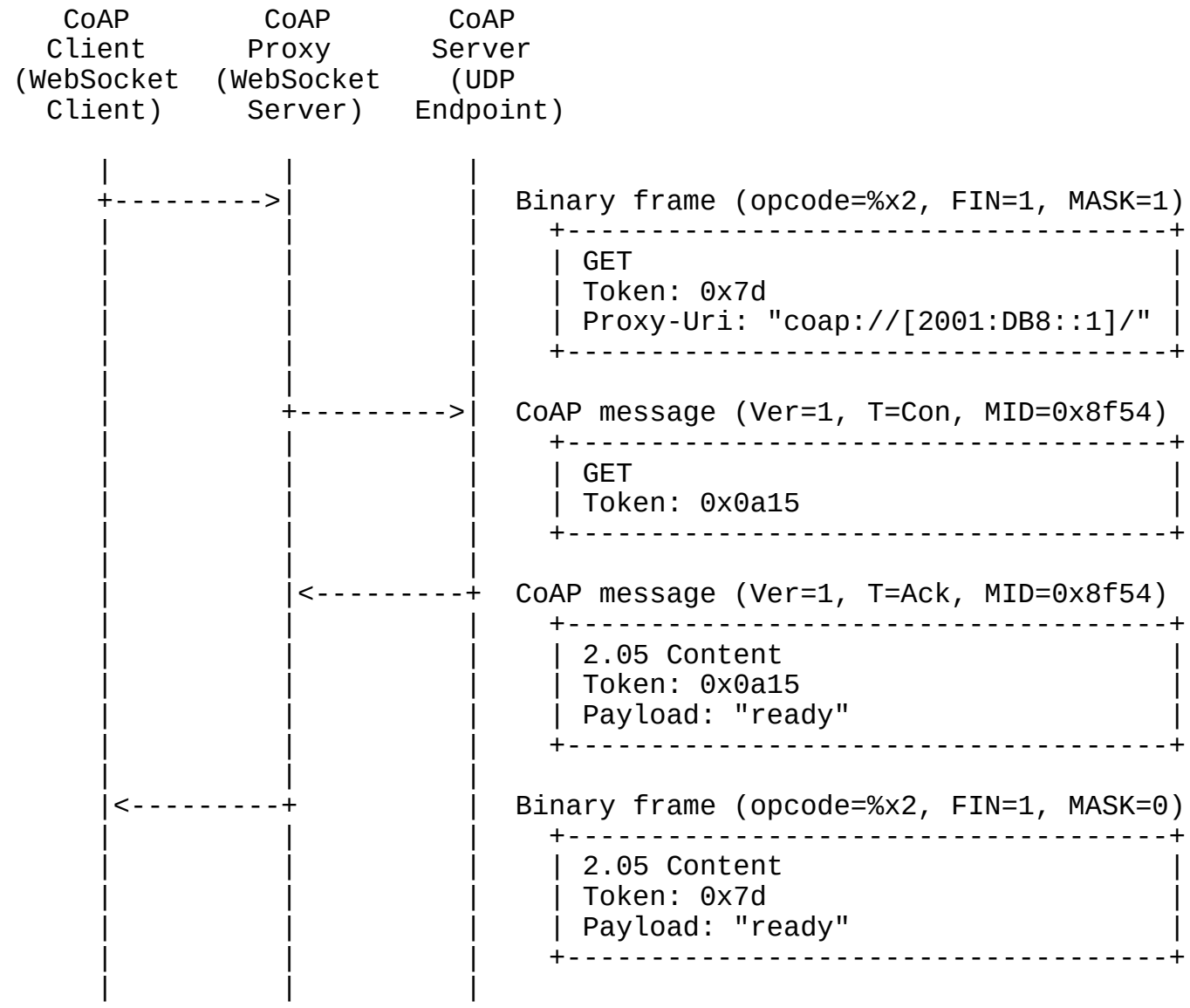

Figure 20: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Appendix C. Change Log

The RFC Editor is requested to remove this section at publication.

C.1. Since draft-core-coap-tcp-tls-02

Merged draft-savolainen-core-coap-websockets-07 Merged draft-bormann-core-block-bert-01 Merged draft-bormann-core-coap-sig-02

Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Christian Groves, Nadir Javed, Michael Koster, Matthias Kovatsch, Achim Kraus, David Navarro, Szymon Sasin, Zach Shelby, Andrew Summers, Julien Vermillard, and Gengyu Wei for their feedback.

Contributors

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Teemu Savolainen
Nokia Technologies
Hatanpaan valtatie 30
Tampere FI-33100
Finland

Email: teemu.savolainen@nokia.com

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

Brian Raymor (editor)
Microsoft
One Microsoft Way
Redmond 98052
United States of America

Email: brian.raymor@microsoft.com